A NEW CHANNEL ROUTING ALGORITHM

Wan S. Chan
California Institute of Technology
Pasadena, California 91125

TM# 5038

September 1982

ABSTRACT

This paper presents a new algorithm for solving the two-layer channel routing problem with doglegging. Based on a set of intuitive and reasonable heuristics, the algorithm tries to obtain a channel routing configuration with a minimum number of tracks. For every benchmark problem tested, the algorithm gives a routing configuration with the smallest number of tracks reported in the literature.

The research was carried out during the author's tenure as a Hewlett-Packard representative at Caltech Silicon Structures Project.

TABLE OF CONTENTS

I. INTRODUCTION

In this paper we present a new algorithm for solving the two-layer channel routing problem with restricted doglegging, identical to the problem considered in [1,3]. Although the algorithm was developed as part of a silicon compiler [5], it can be applied to any channel routing environment.

Our algorithm is based on the following new improved ideas:

(a) by viewing each two-pins subnet as a basic component, we eliminate the need for the "zone" concept introduced in [1];

(b) at each iteration of our algorithm, we identify the set of subnets to be considered for assignment, based on both the vertical and horizontal constraints rather than on the horizontal constraint alone, as proposed in [1];

(c) to determine the assignment of subnets to tracks, we attach weights onto the edges of the bipartite graph and use the max-min matching [4], a refinement of the maximum cardinality matching [4] used in [1];

(d) by appropriately defining the weight on these edges one can introduce various heuristics into the algorithm. In our algorithm, the weight has been defined to control the growth in the length of the longest path through the vertical constraint graph, which is a lower bound on the number of tracks needed;

(e) instead of starting from a density column and assigning all the subnets on one side of the column before considering any subnet on the other side of the column, our algorithm considers assigning subnets on both sides of the column simultaneously. We also derive an algorithm and a theorem to detect the constraint loops for the bidirectional case, which reduce to those presented in [1] for the unidirectional case.

We tested our algorithm on four channel routing problems taken

from the literature and obtained the same best results reported among various papers [1,3].

The paper is organized as follows: in Sec. II we formulate the two-layer channel routing problem with restricted doglegging; in Sec. III we discuss all the important ideas behind our algorithm, using several examples for illustration; in Sec. IV we present our algorithm in high level pseudo code; in Sec. V we summarize the experimental results we obtain; and in Sec. VI we present additional ideas we have on the channel routing problem.

II. PROBLEM FORMULATION

Consider a rectangular routing channel with one row of pins along each of its sides. For each pin, its x-coordinate is fixed but its y-coordinate is to be determined by the router. A net is a set of two or more pins which are to be electrically connected by the router. As shown in Figure 1a, net 1 consists of two pins, one at the bottom side and the other at the left side of the channel. A net with p pins can be viewed as consisting of (p-1) subnets, each of which connects two neighboring (along x-axis) pins of the net. As shown in Figure 1b, net 2 which consists of three pins, can be viewed as two subnets 2a and 2b.

The router is to connect all the pins of each net using a set of vertical and horizontal segments. In the case of no doglegging, each net consists of a single horizontal segment, as shown in Figure 1a. By introducing doglegs at various columns, a net may consist of several horizontal segments [1,2,3]. As in [1,3], we consider a restricted form of doglegging in this paper: doglegs of a net can be introduced only at the column where the net has a pin. In other words, each subnet will consist of a single horizontal segment and dogleg can be introduced only at the endpoints of a subnet, as shown in Figure 1b.

Although this restricted doglegging will, in general, use more tracks than the unrestricted doglegging of [2], we adopt it for the following reasons. If it is assumed that the given x-coordinates of the pins are such that there is enough distance between each pair of columns to satisfy the process design rules, then introducing doglegs only at the endpoints of subnets does not require checking for enough distance between the columns to insert a dogleg. In addition, this restricted doglegging allows the conceptual simplification of treating each subnet as a basic component.

The router uses two independent layers for interconnection:

one used for all horizontal segments and the other for all vertical segments. The electrical connectivity between a horizontal segment and a vertical segment is achieved through a contact located at their intersection.

In order to avoid unintended electrical connection, two horizontal segments which belong to two different nets must not superimpose. Or, equivalently, two subnets which overlap horizontally and which belong to different nets must be assigned to different tracks. For example, in Figure 1b, subnets 2b and 3 must be assigned to different tracks. This type of constraint is refered to as horizontal constraint.

Similarly two vertical segments which belong to two different nets must not superimpose. Hence, if two subnets, which belong to different nets, have a pair of endpoints aligned in the same column, then the subnet whose aligned endpoint is to be connected to a pin at the top side must be assigned a track above the track assigned to the subnet whose aligned endpoint is to be connected to a pin at the bottom side. For example, in Figure 1b, subnet 2a must stay above subnet 1, and subnet 3 must stay above subnet 2b. This type of constraint is refered to as vertical constraint.

Note that the vertical constraint is transitive, i.e. if subnet a must stay above subent b, and subnet b above subnet c, then subnet a must stay above subnet c even though subnets a and c may not have aligned endpoints or may not even overlap horizontally. Unless otherwise stated, the term vertical constraint will include both the direct and indirect (via transitive closure) vertical constraints throughout this paper.

Obviously if there is a loop of vertical constraints, then there exist no solution to the channel routing problem. However such a loop can be broken by inserting a new pin into one of the subnets in

the loop, thus splitting the subnet into two subnets which can be independently assigned to different tracks. Hence without loss of generality, we assume that there is no loop of vertical constraints in the given channel routing problem.

A solution to the channel routing problem is to find an assignment of the subnets to the tracks without violating any horizontal or vertical constraints, e.g. Figure 1a and 1b. In other words, viewing a track as a set of subnets, a solution to the channel routing problem is simply a partition of the set of all the subnets, which does not violate the horizontal and vertical constraints.

There are several criteria for evaluating a channel routing configuration [2] but minimizing the number of tracks used, which is proportional to the chip area, is the most important. In this paper we shall present a heuristic algorithm which tries to obtain a channel routing configuration with a minimum number of tracks.

III. UNDERLYING IDEAS

In this section, we will present all the important ideas upon which our channel routing algorithm is based.

A. Density

Given a channel routing problem, the density of the problem is defined as the largest number of different nets crossing a (vertical) column of the channel. For example, the density of the problem in Figure 1 is 2 and that of Figure 2 is 5. As pointed out in Sec. II, in order to satisfy horizontal constraints, all the subnets of a column which belong to different nets must be assigned to different tracks. Therefore the density is a lower bound on the minimum number of tracks needed for a given channel routing problem.

Let a column which has the largest number of different nets crossing be known as a density column. For example, Figure 2 has two density columns next to each other. In our algorithm, we shall first find all the density columns of the given channel routing problem. For reasons that will become obvious later in Sec. III.E, we choose to start with the middle one among all the density columns. We then assign a different track to each subnet in that column which belongs to a different net. For example, in Figure 2, subnets 1,2,3,4 and 5 are each initially assigned to a different track. The reason for starting with a density column is that since we will need at least density number of tracks, we might as well use that many tracks from the beginning.

B. Front Line Subnets (FLS)

After assigning each subnet in the chosen density column to a track, our algorithm enters an iterative loop where at each iteration a set of yet unassigned subnets closest to the tracks is considered for assignment. Such a set of subnets is called the front-line subnets and they are defined below.

Let left (right, respectively) end-point of a track be defined

as the left- (right-, respectively) most end-point of all the subnets assigned to the track. Let lx(b) (rx(b), respectively) denote the x-coordinate of the left (right, respectively) end-point of b where b can be either a subnet or a track. For the sake of brevity, we will discuss only the subnets on the right of the tracks. All the ideas below apply similarly to the subnets on the left of the tracks, with obvious modification.

To find the rfls (right-front-line-subnets), we first sort all the right subnets into ascending order of their lx. Following that order, we examine each subnet to see if it is a rfls. A right subnet is said to be a rfls if and only if it has a horizontal or vertical constraint with every one of the rfls selected thus far. Intuitively, at each iteration, the set of rfls is the maximal set of right subnets which are at the front line facing the tracks and should be considered for assignment without waiting until the next iteration.

Consider the example in Figure 2. Suppose we first choose the right density column and initially assign subnets 1-5 to tracks 1-5 respectively. Note that all the subnets have been sorted into ascending order of their lx. At the first iteration, the first right subnet considered is subnet 6. Since at this point no rfls has been selected, subnet 6 satisfies all the requirements of a rfls and is selected as a rfls. Next, subnet 7 is considered and since it does not have any horizontal or vertical constraint with subnet 6, subnet 7 is not a rfls. Similarly, subnets 8-12 do not qualify as rfls at this iteration. Intuitively, subnets 7-12, not having any constraint with subnet 6, can be assigned to the same track as subnet 6, and hence should wait until subnet 6 is assigned before they are considered for assignment. In other words, subnet 6 shields subnets 7-12 from the tracks so that they are not at the front line facing the tracks and hence need not be considered for assignment at this iteration. Now suppose subnet 6 is

assigned to track 4 as shown in Figure 2. At the second iteration, the first right subnet considered is subnet 7 and hence it is a rfls. Next subnet 8 is considered. Although it does not have horizontal overlaps with subnet 7, it does have a vertical constraint with subnet 7. Thus subnet 8 is also a rfls at this iteration. However subnets 9-11 are not rfls because they do not have any constraint with subnet 7. Subnet 12 is a rfls because it has vertical constraint with both subnets 7 (via transitive closure) and 8.

The above example illustrates an important idea: by considering the vertical constraint in addition to the horizontal constraint in the definition of fls, we enlarge the set of subnets which ought to be considered at each iteration (e.g. subnet 12 as a rfls). This increase in the scope of consideration will only help to reduce the chances of requiring extra tracks because rfls were unwisely assigned in earlier iterations by arbitrarily choosing one out of several possible assignments.

C. Max-min Matching

After we have chosen the set of fls at each iteration, we must decide how to assign these fls to the tracks so as to minimize the number of tracks required. Following [1], we view this problem as a bipartite matching problem [4, Chap. 5]: each track corresponds to a source node on the left-hand-side of the bipartite graph; each fls corresponds to a destination node on the right-hand-side; and there is an edge between a track node and a fls node if and only if the fls can be assigned to the track without violating any horizontal or vertical constraint. A matching of a bipartite graph is defined as a subset of the edges where no two edges in it are incident to the same node. Since every fls not in a matching given by solving the bipartite matching problem will require a new track, it is clear that we must try to find the maximum cardinality matching [4,1], i.e. the matching with the

largest number of the fls matched to the tracks.

In general there are several maximum cardinality matchings for a given matching problem. In order to make an intelligent choice among these maximum cardinality matchings, we introduce the weight onto the edges of the bipartite graph. This serves as a convenient mechanism for incorporating heuristics into our algorithm by an appropriate definition of the weight. In the following we shall define the weight we use in our algorithm and explain the heuristics behind it.

At each iteration of our algorithm, we define a vertical constraint graph [1] based on the state of the channel routing problem at that iteration, as follows. A vertical constraint graph is a directed graph where each node corresponds to a track or an unassigned subnet, and each arc (u,v) denotes node u must stay above node v due to a direct vertical constraint. The indirect vertical constraint between two nodes, which arises from transitive closure, corresponds to a directed path between these two nodes in the graph. Figures 3a and 3b show a channel routing problem and its vertical constraint graph. Note that a vertical constraint graph must be acyclic, i.e. free of constraint loops, for a channel routing solution to exist. (Throughout this paper, the term edge is used to refer to the undirected edge in the bipartite graph, and the term arc is used to refer to the directed edge in the vertical constraint graph.)

Let the length of a directed path be defined as the number of nodes in that path, or equivalently, 1 plus the number of arcs in that path. Let the critical path length of a node u in a vertical constraint graph, denoted by cpl(u), be defined as the length of the longest path through u in the vertical constraint graph. Let the critical path length of a vertical constraint graph, denoted by cpl, be defined as the length of the longest path through the graph. In other words,

$$cpl = \max \{cpl(u) \mid \text{all nodes } u \text{ in the graph}\} \qquad (1)$$

In Figure 3b, cpl=3.

Since each node in a directed path has a vertical constraint with all other nodes in the path, it must be assigned to a track different from all other nodes in the path. Hence the minimum number of tracks required in a channel routing problem cannot be smaller than its cpl. In other words, cpl is a lower bound on the minimum number of tracks needed for a given channel routing problem. In Figure 3a, density (which is another lower bound on the number of tracks required) is 2, cpl is 3, and it is easy to see that at least 3 tracks are needed to satisfy the vertical constraints among the subnets.

However, unlike density, which is a fixed quantity for a given channel routing problem, cpl tends to increase as the subnets are assigned to the tracks while solving the problem. It is because after a subnet is assigned to a track, the resultant new track will have all the vertical constraints of both the old track and the assigned subnet since they must remain together on the same horizontal position hereafter. In terms of vertical constraint graphs, the resultant graph can be derived from the original graph by moving all the constraint arcs incident with the subnet onto its assigned track, and then deleting the isolated subnet.

Consider the example in Figure 4. Its density=2 as shown in Figure 4a, and before any subnet is assigned, its cpl=2 as shown in Figure 4b. Suppose subnets 1 and 2 are first assigned to tracks 1 and 2 respectively. At the first iteration, subnets 3 and 4 are the fls whereas subnet 5 is shielded by subnet 4. Suppose subnets 3 and 4 are assigned to tracks 1 and 2 respectively as shown in Figure 4c. Then the cpl of the resultant graph is increased to 3, a number greater than its density, as shown in Figure 4d (where we use u+v to denote the track which consists of subnets u and v). On the other hand, if subnets 3 and 4 are assigned to tracks 2 and 1 respectively, then the cpl remains at

2 and, as shown in Figure 4e, only 2 tracks are needed to solve the given channel routing problem. Thus there is a need to choose among the maximum cardinality matchings the one which minimizes the increase in cpl.

Based on this observation, we incorporate the following heuristics into our algorithm. Let the weight W(u,v) of an edge (u,v) between track u and subnet v be defined as -1 times the cpl through the track resulting from assigning subnet v into track u.

$$W(u,v) = -1 * cpl(u+v) \qquad (2)$$

From (1) and (2) we have,

cpl of the resulting vertical constraint graph

GEQ max {cpl(u+v) | all (u,v) in the matching}

= max {-1*W(u,v) | all (u,v) in the matching}

= -1 * min{W(u,v) | all (u,v) in the matching}

Hence in order to keep the cpl of the resulting vertical constraint graph minimum, our problem becomes that of finding a maximum cardinality matching for which the minimum of the weights of the edges in the matching is maximum, i.e. the max-min matching problem [4]. We use the Threshold Method [4, pp. 198] to find the max-min matching.

The term cpl(u+v) can easily be computed as follows. Given a vertical constraint graph, let us assign the top-down level numbers (tdl) to the nodes as follows:

Top-down level assignment algorithm:

```
BEGIN
   i:=0;
   WHILE not all nodes are assigned a tdl, DO
   BEGIN
      i:=i+1;
      FOR each node u with no incoming arc, DO
      BEGIN
         set tdl(u):=i;
         delete u and all the arcs incident to it, from the graph
      END
   END
END
```

In other words, tdl(u) is the length of the longest path from the top to node u. Similarly, we define the bottom-up level number (bul) of a node. Hence

$$cpl(u) = tdl(u) + bul(u) - 1 \qquad (3)$$

Since the longest path from the top to node u+v, resulting from assigning node v into node u, is simply the longer of the two longest paths from the top to node u and to node v,

$$tdl(u+v) = \max \{ tdl(u) , tdl(v) \} \qquad (4)$$

Similarly,

$$bul(u+v) = \max \{ bul(u) , bul(v) \} \qquad (5)$$

Applying (3) to u+v and substituting (4) and (5), we have

$$cpl(u+v) = \max\{tdl(u),tdl(v)\}+\max\{bul(u),bul(v)\}-1 \qquad (6)$$

We can further refine the definition of the weight by noting that if there is an edge (u,v), and if rx(u)=lx(v), then the right-most subnet assigned to track u and subnet v belongs to the same net (because the existence of edge (u,v) implies that u and v do not have vertical constraint and with rx(u)=lx(v), we can conclude that they

share the pin located at rx(u)). By adding 0.5 to the weight of such (u,v) to encourage (u,v) from being chosen in the max-min matching, we will tend to reduce unnecessary doglegging.

D. Delay Assignment

At each iteration of our algorithm, after finding the max-min matching between the tracks and the fls, we assign the fls to the tracks according to the max-min matching. However not all fls need to be assigned at this point for the problem solving process to progress, i.e. to introduce new fls from the remaining subnets. Since the assignment of the remaining subnets depends critically on the past assignments, intuitively it is a good strategy to avoid any premature assignment, (i.e. commiting a subnet to a track before it is necessary for the introduction of new fls,) and thus retain as much flexibility as possible. The following example illustrates the advantage of such a delay assignment strategy.

Consider the channel routing problem in Figure 5a which has density=3. Suppose subnets 1,2 and 3, which intersect a density column, are initially assigned to tracks 1,2 and 3 respectively. At the first iteration, we have the vertical constraint graph as shown in Figure 5b, and the weighted bipartite graph as shown in Figure 5c. Suppose the max-min matching obtained is {(1,4),(2,5)}. If we assign both subnets 4 and 5 accordingly, we get the vertical constraint graph shown in Figure 5d and the weighted bipartite graph shown in Figure 5e. Clearly only one of subnets 6 and 7 can be assigned to track 1+4, and a new track is needed for the other subnet. Observe that subnets 6 and 7 are shielded by subnet 4 only and after subnet 4 is assigned to a track, subnets 5, 6 and 7 form a legitimate set of fls. If at the end of the first iteration we assign only subnet 4 to track 1, then we get the vertical constraint graph in Figure 5f and the weighted bipartite graph in Figure 5g. There is a max-min matching {(3,5),(1+4,6),(2,7)} which

matches all the fls and hence does not require any new track.

Therefore the delay assignement strategy is incorporated into our algorithm as follows. After finding a max-min matching, our algorithm assigns to the tracks only those fls which are to the left of the first subnet not yet chosen as a fls (recall that those unchosen subnets are sorted in ascending order of their lx). In other words, only those fls with rx<lx of the left-most non-fls subnet are assigned, all other fls remain as fls for the next iteration, and new fls are chosen in the same manner as presented in Sec. III.B. The delay assignment strategy is also incorporated in Algorithm #2 of [1] where at Step 4 only nets terminating at the current zone are assigned.

E. Bidirectional Search

Thus far we have presented our ideas and examples for the case with no subnet on the left of the chosen density column. Since a density column can be at any x-coordinate of the channel, we must consider the general case where there are subnets on both sides of the chosen density column. One obvious approach, taken in [1], is to consider one side of the chosen density column first, iteratively assign all the subnets on that side, then turn around and iteratively assign the subnet on the other side of the chosen density column. Our experience with this approach indicates that because the second side is completely ignored while assigning the subnets on the first side, it tends to require more tracks when assigning the subnets on the second side. Therefore we adopt the following bi-directional search strategy: at each iteration we consider both sides of the tracks simultaneously by building a weighted bipartite graph and finding a max-min matching for each side. Since a density column is the column where the largest number of subnets intersect, it tends to be the critical area where new tracks are added. To spread these difficult areas evenly on both sides of the chosen density column, we take the

chosen density column to be the middle one among all the density columns of a given channel routing problem. With this bidirectional strategy, we do get channel routing configurations with smaller number of tracks.

F. Constraint Loops Detection

Although an edge (u,v) in the bipartite graph guarantees that subnet v can be assigned into track u without violating horizontal or vertical constraints, a simultaneous assignment of two or more subnets to the tracks according to a matching may create a vertical constraint loop. To see this point, consider the example in Figure 6a which has density=2. Suppose subnets 1 and 2 are initially assigned to tracks 1 and 2. For the first iteration, the vertical constraint graph and the bipartite graph are shown in Figure 6b and 6c. Since {(2,4),(1,3)} is the only maximum cardinality matching, it is also the only max-min matching. If we make the assignment accordingly, we find that there is a vertical constraint loop between tracks 1+3 and 2+4 as shown in Figure 6d.

In [1], an algorithm was given to remove the edges in a matching which create constraint loop. However that algorithm cannot be adopted into our algorithm because it works only for a unidirectional search and not a bi-directional search. Instead we use the following algorithm to detect constraint loops. The input to the algorithm consists of a matching of the bipartite graph and the associated vertical constraint graph. The nodes and the edges are deleted from the bipartite graph. For the remainder of Sec. III.F, the term nodes refers to the track and fls nodes in the bipartite graph (a subset of those in the vertical constraint graph). The term nabove of a node is defined to be the number of nodes which is above the node in the vertical constraint graph.

Constraint loops detection algorithm:

```
BEGIN
   compute nabove of every node;
   WHILE NOT every node is deleted, DO
   BEGIN
      DO
      BEGIN
      (a)delete every matching edge between nodes with nabove=0;
         (note: if a track has two matching edges, one on each side,
          then all three nodes must have nabove=0 in order to delete)
      (b)delete every node with nabove=0 and with no matching edge;
      (c)decrement nabove of every node which has an arc from each
         node just deleted
      END
      UNTIL no deletion is possible;
      IF every node is deleted THEN done ELSE
      BEGIN
      (d)choose as an EX edge, a matching edge which has a node with
         nabove=0 (its other node must have nabove>0);
      (e)delete the chosen EX edge
      END
   END
END
```

To illustrate the algorithm, consider the matching and the vertical constraint graph shown in Figure 7a and 7b respectively. At the first iteration, no matching edge can be deleted. Although both track 1 and subnet 5 have nabove=0, their matching edge (1,5) cannot be deleted because track 1 is matched to subnet 3 on the other side which has nabove=1. Suppose matching edge (2,4) is chosen over (1,3) as an EX edge and is then deleted. At the second iteration, node 4 is

first deleted which causes node 3 to have nabove=0. Next matching edges (1,5) and (1,3) are deleted since all three nodes now have nabove=0. Since there are no matching edges left, all the remaining nodes are deleted in the order of their top-down level: 1,3,5 and 2.

The following theorem guarantees that a matching without any EX edge will not result in a constraint loop after the assignment is made according to the matching. Thus for each set of fls, we iterate on the process of finding the max-min matching of the new bipartite graph, and remove all its EX edges from the bipartite graph, until a max-min matching with no EX edge is found.

THEOREM

A matching has no EX edge if and only if the subnet-to-track assignment made according to the matching will not result in a constraint loop.

PROOF:

We will first prove that if there is no EX edge there is no constraint loop. Since a constraint loop must consist of at least two nodes (resulting from the assignment of two fls nodes to two tracks nodes) which have mutual vertical constraint on each other, these resultant nodes must both have nabove>0. Since every node of the matching edges deleted in Step (a) has nabove=0, the node resulting from the assignment based on these deleted edges will have nabove=0 and hence cannot be part of a constraint loop. Since every node deleted in Step (b) will only result in a node with nabove=0 after the assignment, they can be forgotten as far as the possibility of creating constraint loops is concerned. Applying the same reasoning in the subsequent iterations of Steps (a) and (b) until all nodes are deleted, we conclude that no EX edge implies no constraint loop.

Next we will prove that if there is an EX edge then there is a constraint loop. Let (u(1),v(1)) be an EX edge. Without loss of

generality, assume that u(1) has nabove=0, and v(1) has nabove>0. Since the vertical constraint graph before assignment is acyclic, there exists a node u(2) with nabove=0 which is an ancestor of v(1). Since u(2) is not yet deleted, there exists a matching edge (u(2),v(2)) where v(2) has nabove>0. Applying the same reasoning repeatedly, we have an infinite sequence of matching edges (u(1),v(1)),(u(2),v(2)),... Since the graph is finite and the sequence is not, there exists an k such that u(1)=u(k) and v(1)=v(k). Since u(i+1) is an ancestor of v(i), after assigning according to the matching, u(i+1)+v(i+1) is an ancestor of u(i)+v(i). Hence there is a constraint loop around u(1)+v(1) after the assignment.

Q.E.D.

Observe that our constraint loop detection algorithm and the theorem reduce to those in [1] for the unidirectional case.

IV. ALGORITHM

Combining the ideas presented in the previous section, we arrive at the following heuristic algorithm for solving a two-layer channel routing problem with a minimum number of tracks.

Channel routing algorithm:

```
BEGIN
   initialize each subnet in the middle density column as a track;
   WHILE a subnet is not yet assigned, DO
   BEGIN
      find left and right sets of fls;
      build left and right weighted bipartite graphs;
      DO
      BEGIN
         delete EX edges of last matchings from bipartite graphs;
         find left and right max-min matchings of new bipartite graphs;
      END
      UNTIL the new matchings have no EX;
      determine which fls should be assigned in order to get new fls;
      FOR each of these fls, DO
      BEGIN
         IF it is in the matching, THEN assign to track accordingly
                                  ELSE create a new track for it
      END
   END;
   position the tracks to satisfy vertical constraints among them
END.
```

V. EXPERIMENTAL RESULTS

The algorithm presented in the previous section has been implemented in the programming language MAINSAIL on a DEC 20 computer and a HP 9826 desk-top computer. The algorithm was tested on four channel routing problems taken from the literature. They range in size and complexity and the last one is the well-known "Difficult Example" [1,3]. For each problem our algorithm uses the same number of tracks as the minimum reported among various papers. Table 1 summarizes for each benchmark problem, its source, the number of subnets it has, its density, the minimum number of required tracks reported in the literature along with its reference, and the number of tracks used by our algorithm. Note that our algorithm uses only the density number of tracks for the first two problems, and one plus density for the other two. The channel routing configuration obtained by our algorithm are presented in Figures 2,8,9 and 10.

In order to determine the effectiveness of our algorithm, it should be tested against many more benchmark channel routing problems. However they are not readily available in the machine readable form and to code them by hand is a tedious and error-prone process.

VI. ADDITIONAL COMMENTS

A. Wire Length

In addition to the number of tracks used, the total length of the wire is also an important criterion in evaluating a channel routing configuration. Given a channel routing problem, the total length of its horizontal segments is fixed. Given a channel routing configuration, the length of a vertical segment can be reduced in two ways: (a) by reassigning a subnet which causes a dogleg to a new track to reduce or even eliminate the length of the dogleg; (b) by reassigning a subnet, which is not a subnet with one pin at the top side and another at the bottom side of the channel, to a new track closer to the (top or bottom) side where the pins are. Of course, the new assignment must still satisfy the vertical and horizontal constraints. For example in Figure 8, we can reassign subnet 1 from track 8 to track 6 to *eliminate* its dogleg of length=2, and reassign subnet 2 from track 7 to track 11 to reduce wire length by 2*(11-7).

B. Contacts

Another important criterion to evaluate a channel routing configuraton is the number of contacts, or vias, used to electrically connect the two independent interconnect layers. Each pin along the top or bottom sides of a channel needs one contact to connect to its horizontal segment(s) if there is no dogleg, and it needs two contacts if there is a dogleg in that column. Besides, for a vertical segment which is not intersected by any horizontal segment, it can be routed in the layer used for horizontal segments and thus will not need a contact between itself and its horizontal segment(s). Hence we have

number of contacts required
= (number of pins on the top and bottom sides of the channel) +
(number of doglegs) - (number of vertical segments which are not intersected by any horizontal segments) (7)

Given a channel routing problem the number of pins in the top and bottom sides are fixed. However given a routing configuration, one can reassign the subnets to new tracks, within the restriction imposed by the vertical and horizontal constraints, to decrease the number of doglegs and to increase the number of vertical segments not intersected by any horizontal segments.

Therefore by post-processing a channel routing configuration, one can decrease the wire length and the number of contacts required.

C. Three-layer Channel Routing

With the recent advance in the process technology, in particular the second metal layer, it is possible to have three independent layers for interconnection. For such process technology the channel routing problem becomes relatively simple. The three-layer channel routing problem can be formulated as in Sec. II except that the first layer is used for the horizontal segments, the second layer for the vertical segments connected to pins along the top side of the channel, and the third layer for the vertical segments connected to pins along the bottom. Since the second and the third layers can superimpose without causing any electrical connection, one no longer needs to be concerned about the vertical constraints. Hence the three-layers channel routing problem becomes that of finding the minimum cardinality set of paths through the horizontal constraint graph, where each node corresponds to a subnet and each edge (u,v) indicates that node v is to the left of node u, i.e. v can be assigned to the same track as u. A solution to this problem can be found by using the greedy strategy, i.e. assign each subnet to the track closest to it. It is also easy to show that for any three-layer channel routing problem there always exists an optimal solution which uses the *same number of* tracks as its density.

VII. CONCLUSION

We have presented a new algorithm for solving the two-layer channel routing problem along with the insight and the experimental results. It will be used as part of a general routing system now being implemented based on the ideas developed by Mr. G. Clow and myself. Although our general routing system was initially intended as part of a silicon compiler [5], our ideas apply just as well in an interactive graphic computer-aided design environment.

ACKNOWLEDGEMENT

This work was carried out during the author's tenure as a visiting associate representing Hewlett-Packard at the Silicon Structures Project in California Institute of Technology. The author wishes to express his sincere gratitude to HP and SSP for providing an excellent research environment.

REFERENCES:

[1] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, vol. CAD-1, pp. 25-35, January 1982.

[2] R. L. Rivest and C. M. Fiduccia, "A greedy channel router," Proceedings of 19th Design Automation Conference, paper 27.2, June 1982.

[3] D. N. Deutsch, "A dogleg channel router," Proceedings of 13th Design Automation Conference, pp. 425-433, 1976.

[4] E. L. Lawler, "Combinatorial optimization: networks and matroids," Holt, Rinehart and Winston, New York, 1976.

[5] T. S. Hedges, K. H. Slater, G. W. Clow and T. Whitney, "The SICLOPS silicon compiler," to appear in Proceedings of International Circuit and Computer Conference, September 1982.

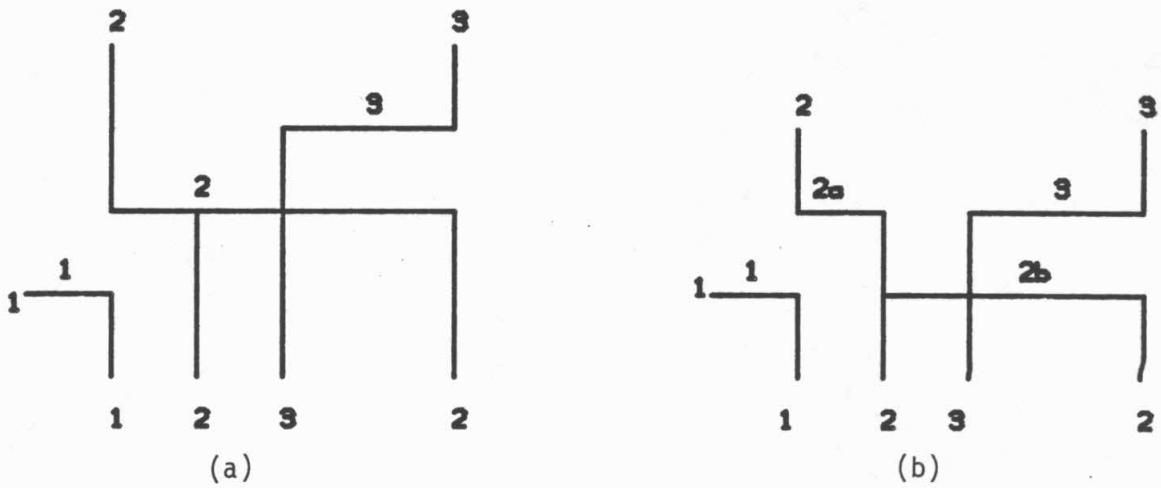


Figure 1.

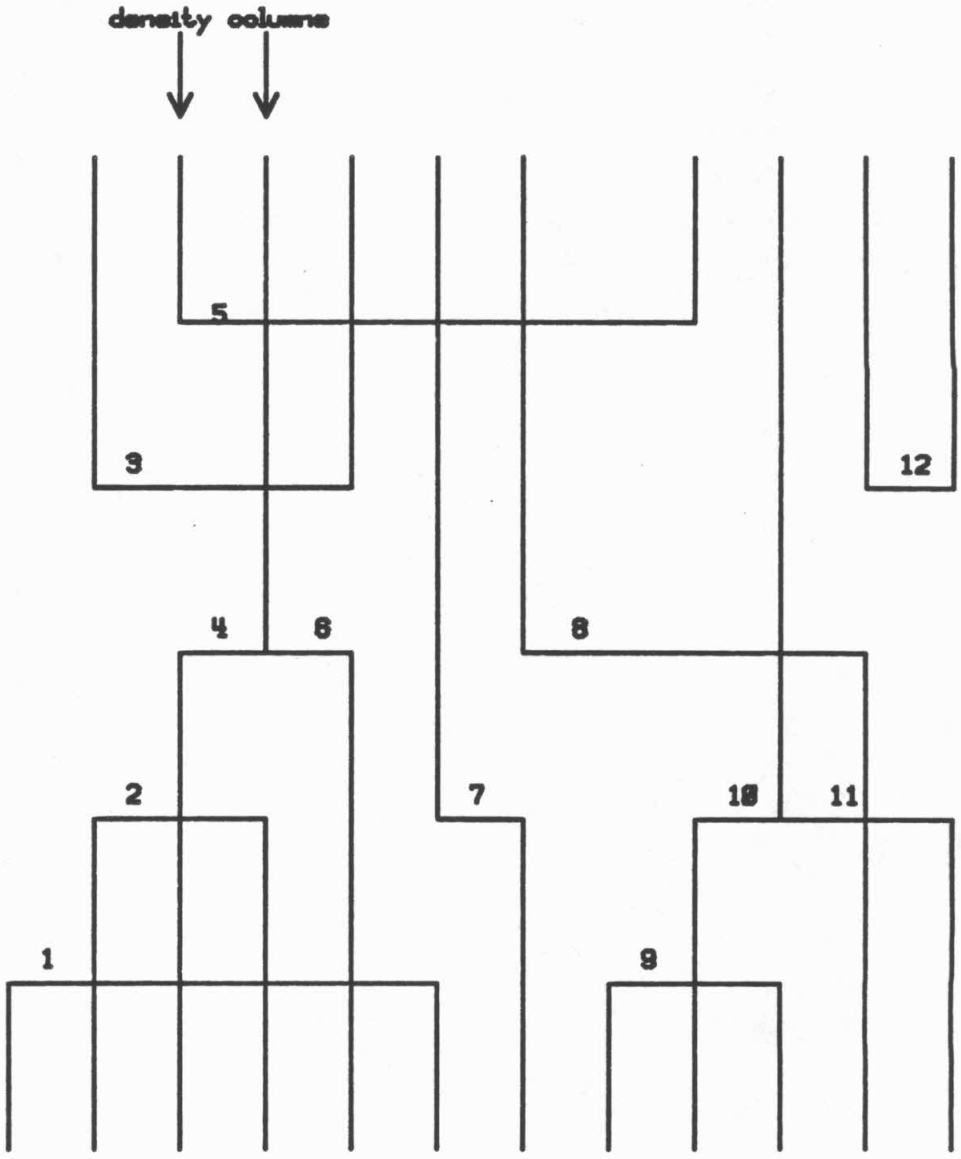


Figure 2.

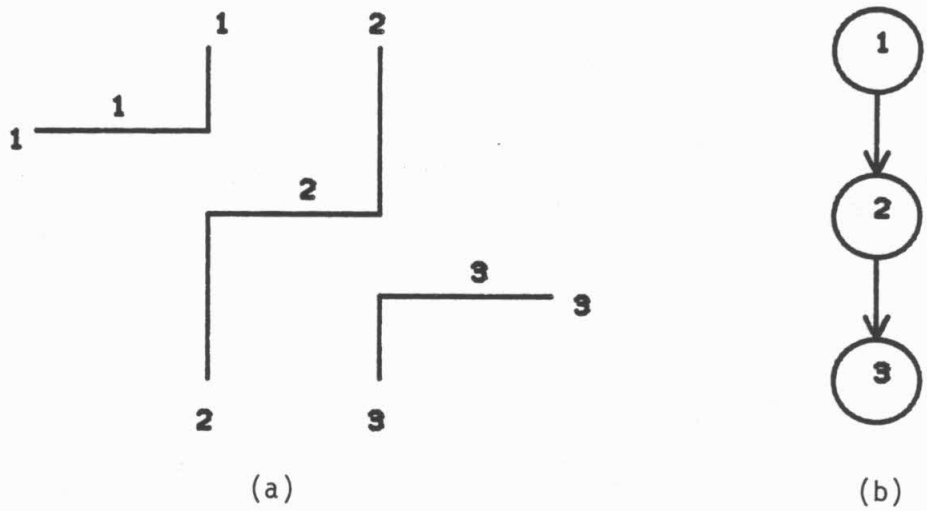
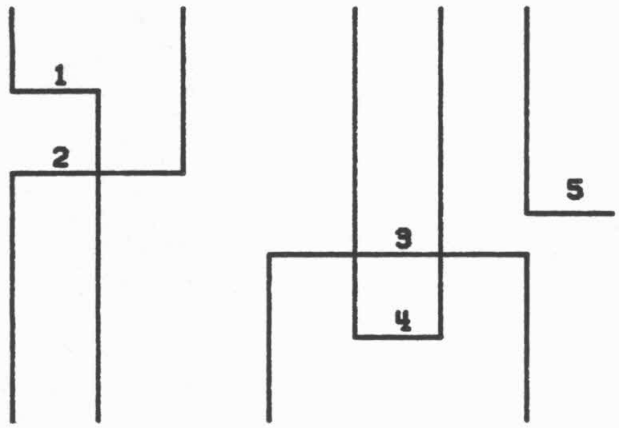
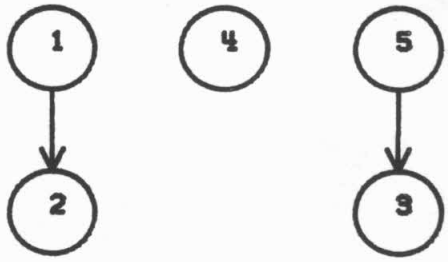


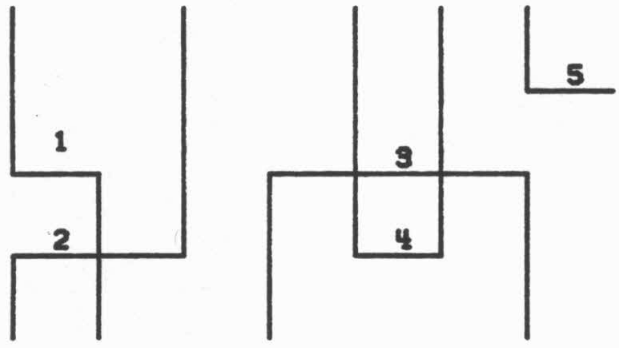
Figure 3.



(a)

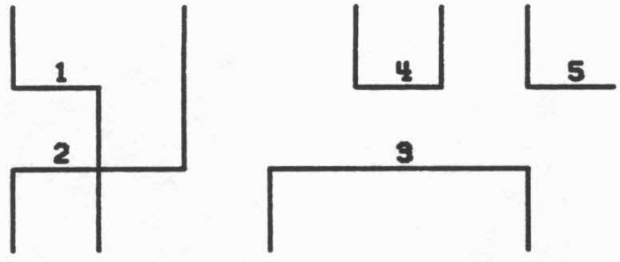


(b)



(c)

(d)



(e)

(f)

Figure 4.

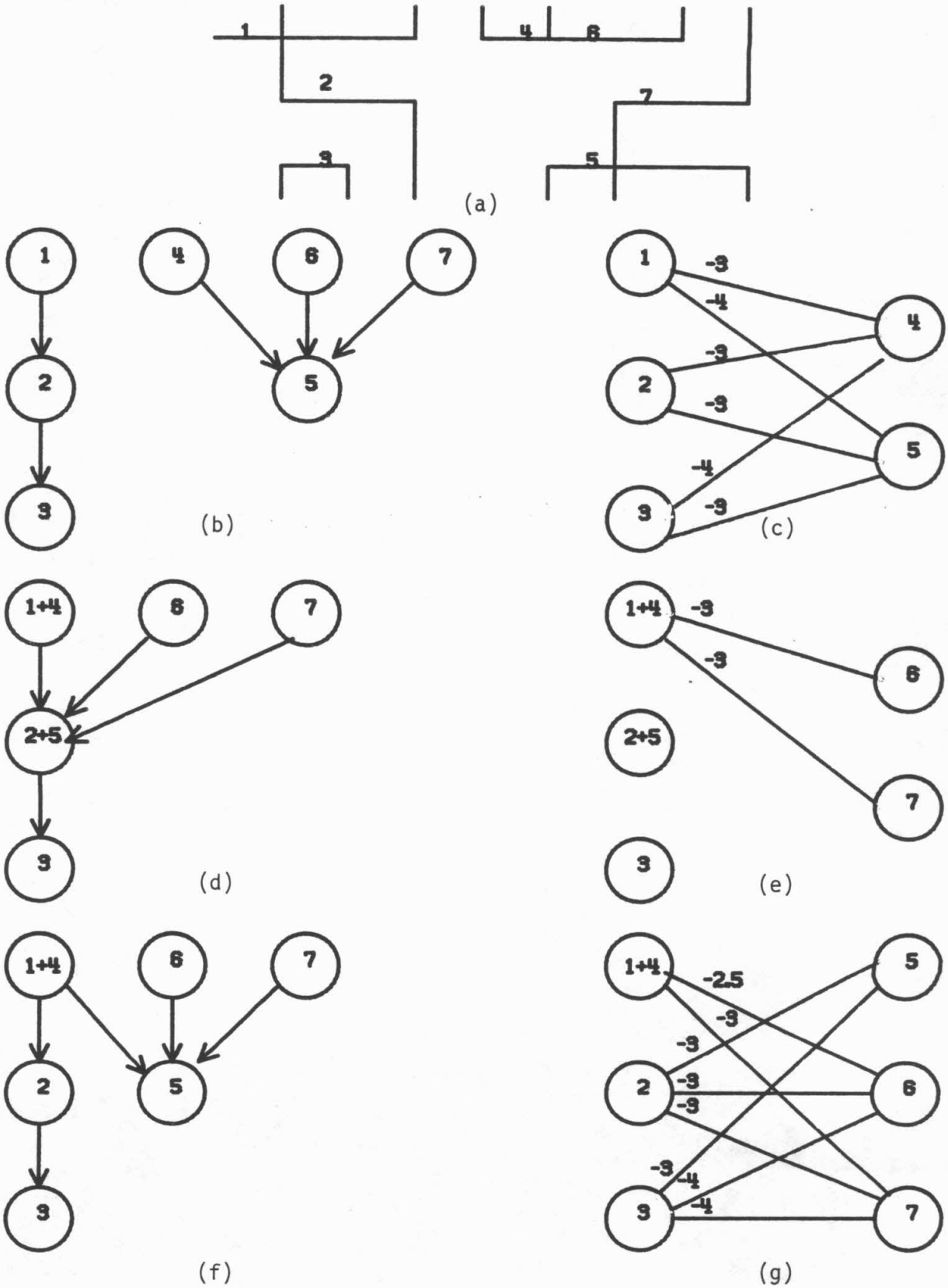
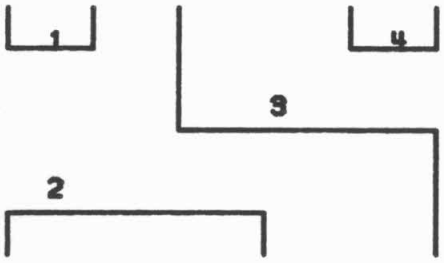
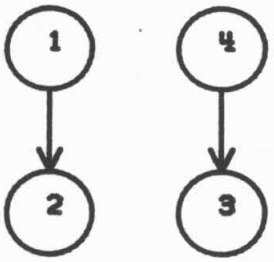


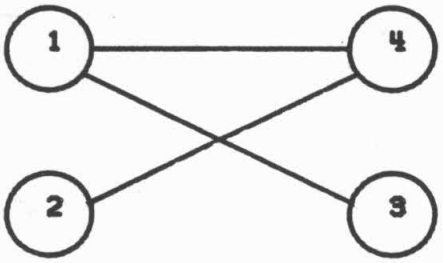
Figure 5.



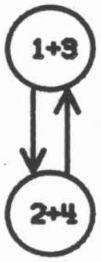
(a)



(b)



(c)



(d)

Figure 6.

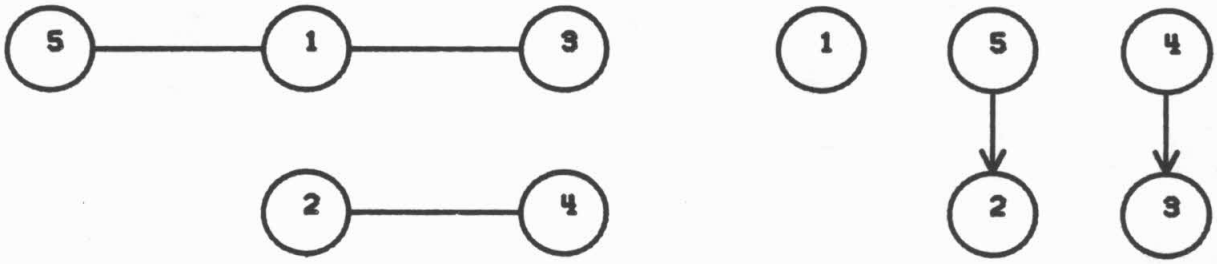


Figure 7.

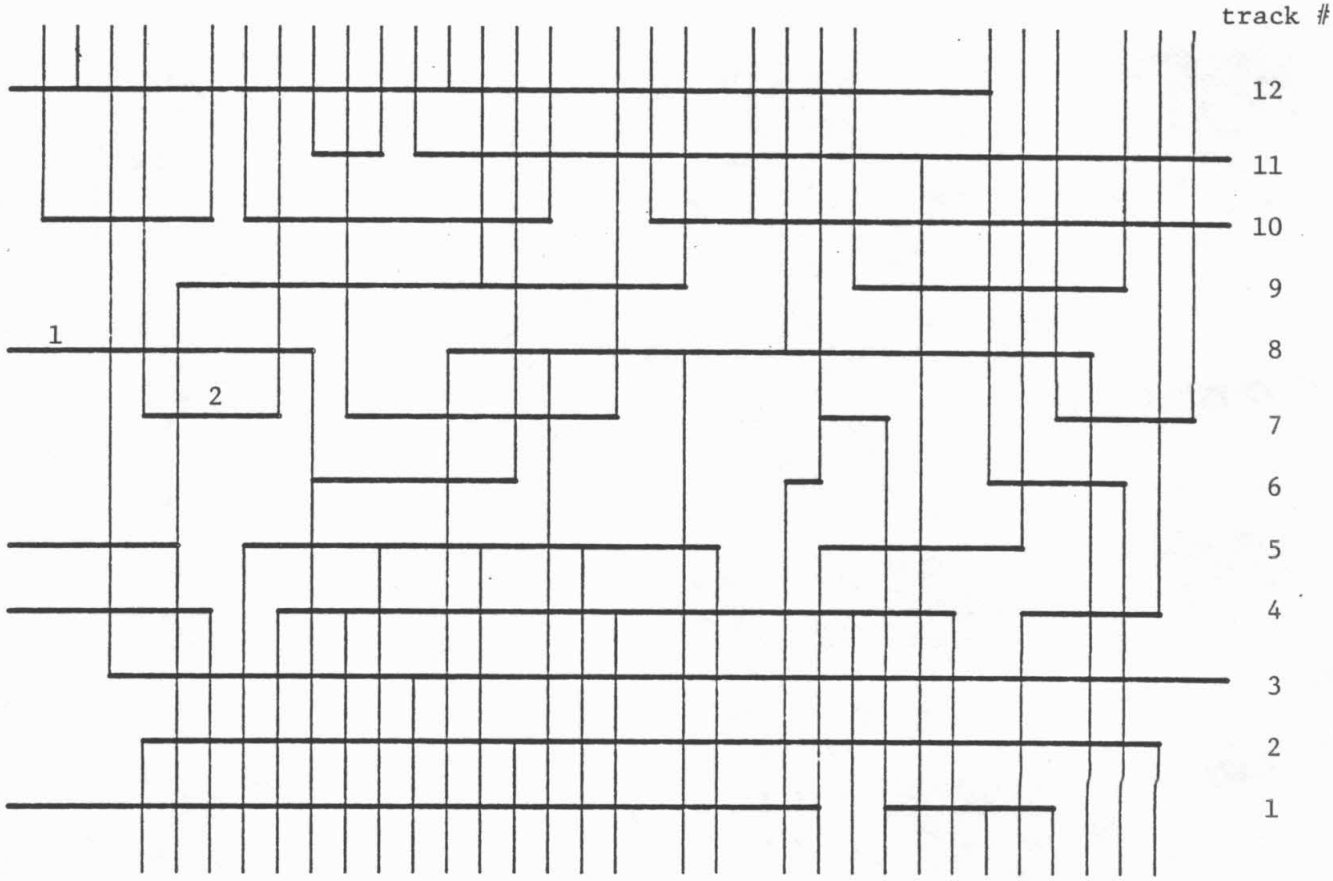


Figure 8.

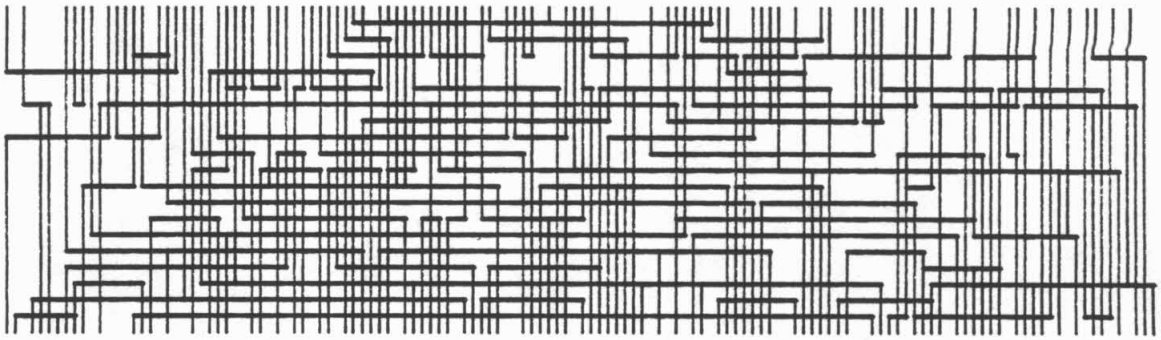


Figure 9.

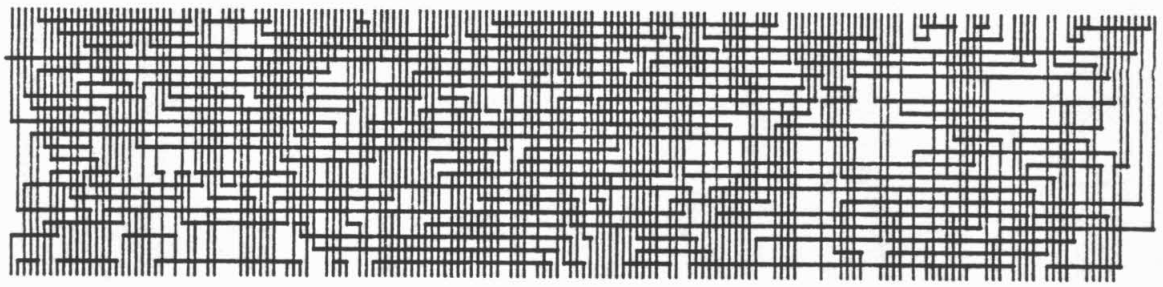


Figure 10.

benchmark problem	# of subnets	density	best result reported	our result
1. [1. Figure 1]	12	5	5 [1]	5
2. [1. Figure 25]	44	12	12 [1]	12
3. [3. Figure 11]	130	18	19 [3]	19
4. [3. Figure 10]	218	19	20 [1]	20

Table 1.

CAPTIONS

Figure 1. Minimum tracks routing configurations,
(a) without doglegging;
(b) with restricted doglegging.

Figure 2. A simple channel routing problem.

Figure 3. An example of cpl>density,
(a) routing configuration;
(b) vertical constraint graph.

Figure 4. Controlling the growth in cpl,
(a) the given channel routing problem;
(b) initial vertical constraint graph;
(c) final routing configuration if 3 and 4 are assigned to 1 and 2 respectively;
(d) final vertical constraint graph if 3 and 4 are assigned to 1 and 2 respectively;
(e) final routing configuration if 3 and 4 are assigned to 2 and 1 respectively;
(f) final vertical constraint graph if 3 and 4 are assigned to 2 and 1 respectively.

Figure 5. Advantage of delay assignment strategy,
(a) the given channel routing problem;
(b) initial vertical constraint graph;
(c) initial weighted bipartite graph;
(d) vertical constraint graph if 4 and 5 are assigned to 1 and 2 respectively;
(e) bipartite graph if 4 and 5 are assigned to 1 and 2 respectively;
(f) vertical constraint graph if only 4 is assigned to 1;
(g) bipartite graph if only 4 is assigned to 1.

Figure 6. Constraint loops created by simultaneous assignment,
(a) the given channel routing problem;
(b) initial vertical constraint graph;
(c) initial bipartite graph;
(d) vertical constraint graph after 4 and 3 are assigned to 1 and 2 respectively.

Figure 7. An example for constraint loop detection algorithm,
(a) bidirectional bipartite graph;
(b) vertical constraint graph.

Figure 8. Routing configuration generated for benchmark problem #2.

Figure 9. Routing configuration generated for benchmark problem #3.

Figure 10. Routing configuration generated for benchmark problem #4.

Table 1. Summary of experimental results.